МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики  
и прикладных информационных технологий

ЛАБОРАТОРНАЯ РАБОТА №2

ЧИСЛЕННОЕ РЕШЕНИЕ СТАЦИОНАРНОГО

УРАВНЕНИЯ ШРЁДИНГЕРА:

ТЕОРИЯ ВОЗМУЩЕНИЙ

Направление 01.04.02 Прикладная математика и информатика

Профиль Математическое моделирование и вычислительная математика

|  |  |  |  |
| --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м.н., пр. | А.И. Шашкин \_\_.\_\_.2023 |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | И.Б. Рахимов |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_ | д.ф.-м.н., пр. | Ю.К. Тимошенко |

Воронеж 2023

Содержание

Цели и задачи 3

1. Одномерное стационарное уравнение Шрёдингера. Теория возмущений 5

1.1. Частица в одномерной потенциальной яме 8

2. Программная реализация 9

3. Результаты численных экспериментов и их анализ 10

Список использованных источников 12

Приложения 13

Приложение 1 Компьютерный код 13

# Цели и задачи

### Цели работы

Целями лабораторной работы являются практическое освоение информации, полученной при изучении курса «Компьютерное моделирование в математической физике» по теме «Численное решение стационарного уравнения Шрёдингера», а также развития алгоритмического мышления и приобретения опыта использования знаний и навыков по математике, численным методам и программированию для решения прикладных задач физико–технического характера.

### Задачи работы

**Формулировка проблемы:** электрон находится в потенциальном поле (невозмущенная система) (рис. 1):

где , , .

С помощью теории возмущений найти:

* энергию основного состояния, с учётом поправок до второго порядка включительно;
* волновую функцию основного состояния, с учётом поправок первого порядка.

Возмущённую систему смоделировать самостоятельно, создав в потенциальной функции из лабораторной работы №1 пик произвольной формы. Сравнить результаты с данными, полученными методом пристрелки.

Для потенциальной функции возмущённой системы выбран пик следующего вида (Рис. 2):

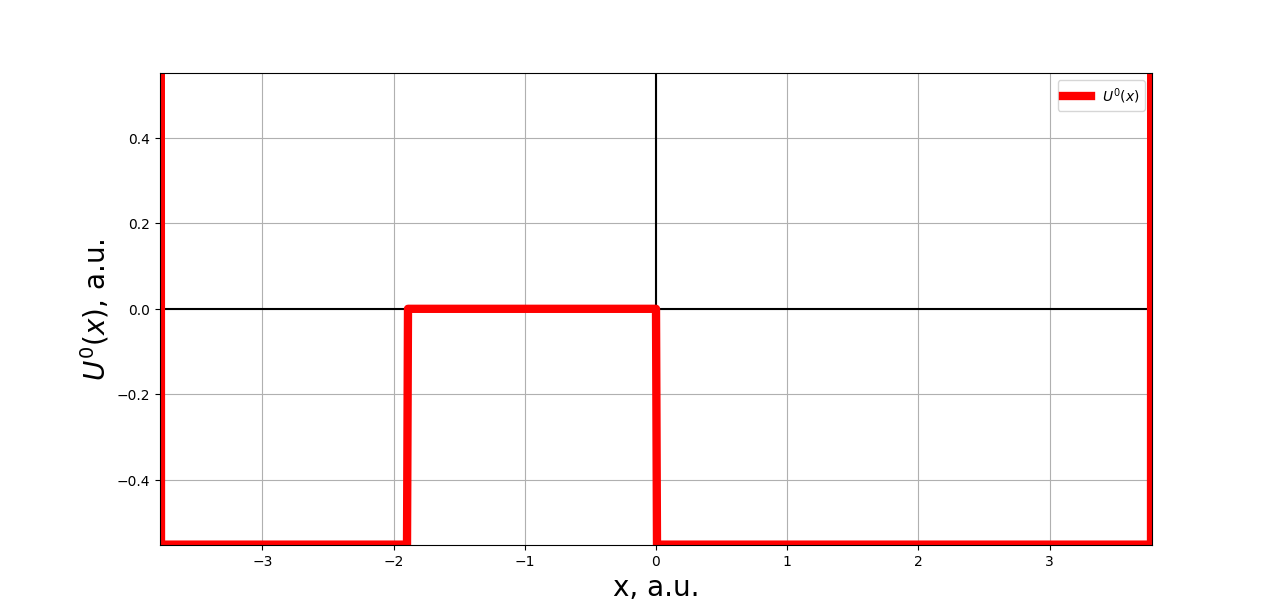


Рис. 1. График невозмущенной потенциальной функции

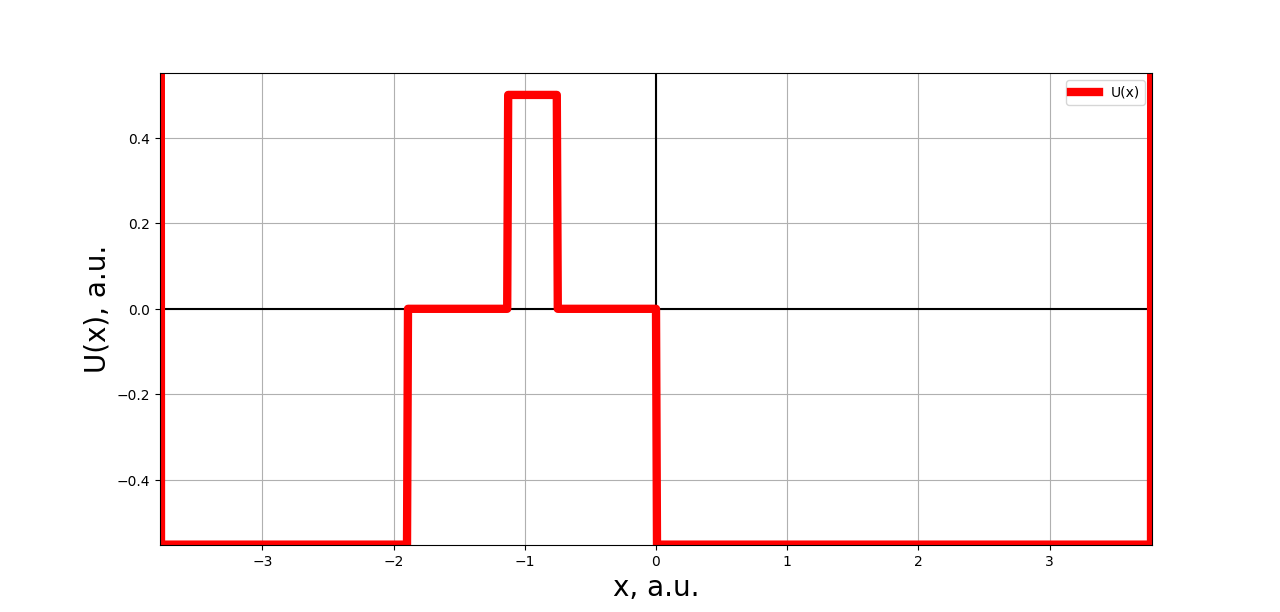


Рис. 2. График возмущенной потенциальной функции

# Одномерное стационарное уравнение Шрёдингера. Теория возмущений

В данном отчете не будет подробно описана математическая часть одномерного стационарного уравнение Шрёдингера, так как это уже было сделано в отчете по первой лабораторной работе, вместо этого здесь будет подробно разобрано его решение в рамках теории возмущений.

К числу наиболее распространённых у физиков приближенных методов вычисления собственных значений и собственных функций оператора Гамильтона относится метод стационарных возмущений Релея-Шрёдингера, так же называемый методом или теорий возмущений. В рамках этого подхода предполагается что оператор Гамильтона, чьи собственные значения и собственные функции требуется определить, может быть представлен в виде:

где – гамильтониан идеализированной задачи, решение которой можно найти либо аналитически, либо относительно простым численным путём, – называется оператором возмущения, или просто возмущением.

Оператором возмущения может быть либо часть гамильтониана, которая не учитывалась в идеализированной задаче, либо потенциальная энергия, связанная с наличием внешнего воздействия. Существенно, что оператор возмущения должен содержать «малый параметр», который будет обеспечивать сходимость рядов теории возмущений, о которых далее пойдёт речь.

Идеализированную систему, которую описывает гамильтониан , называют невозмущенной системой, а систему с гамильтонианом – возмущенной системой. В рамках теории возмущений удаётся получить формулы, которые определяют энергии и волновые функции стационарных состояний через известные значения энергии и волновых функций невозмущенной системы.

Стационарные уравнения Шрёдингера для невозмущенной и возмущенной систем имеют вид:

Буквой обозначена совокупность всех независимых координат. Задание определяет положение точки в абстрактном пространстве, которое называют конфигурационным пространством.

Для системы, состоящей из одной частицы, конфигурационной пространство совпадает с обычным декартовым пространством , [2].

Для частицы в одномерной пространстве – это одна из декартовы координат, например .

Волновые функции возмущенной и невозмущенной систем ортонормированы:

где символ «» означает комплексное сопряжение, а – символ Кронекера, который определяется следующим образом:

В теории возмущений решение уравнения ищутся в виде рядов:

где , – величины -го порядка малости по возмущению , называемые -ми поправками или поправками -го порядка. Первые слагаемые рядов определяются следующими формулами:

где

Штрих над знаком суммы означает пропуск слагаемого, имеющего равные индексы, то есть .

Интеграл называется матричным элементом оператора по невозмущенным волновым функциям. Первая поправка к собственному значению равняется квантовомеханическому среднему значению возмущения в состоянии , а поправка второго порядка к энергии основного состояния не может быть положительной.

Ряды сходятся если выполняется неравенство:

Во многих случаях для решения задачи достаточно ограничиться вычислением энергии с учетом поправок до второго порядка включительно и волновой функции с учетом поправок первого порядка.

## Частица в одномерной потенциальной яме

Частицу в одномерной потенциальной яме с бесконечными стенками будем рассматривать как невозмущенную систему (рис. 1). Она соответствует потенциальной функции . График возмущённой потенциальной функции представлен соответствует (рис. 2).

Возмущение представляет собой разность двух функций:

Для вычисления энергии и волновой функции возмущенной системы используются собственные значения и собственные функции, найденные численно в первой лабораторной работе методом пристрелки.

В следующей главе будет подробно объяснена программная реализация на языке программирования Python решения одномерного стационарного уравнения Шрёдингера с помощью теории возмущений.

# Программная реализация

В Приложении 1 представлена программа численного решения одномерного стационарного уравнения Шрёдингера. Использовались атомные единицы Хартри. Программа реализована на языке Python 3.10.7 в графической среде разработки «PyCharm Community Edition 2023.2.5», использовался интерпретатор CPython, операционная система Windows 11 Профессиональная.

В строках 1-440 находится реализация метода пристрелки, которая подробно была описана в отчёте по первой лабораторной работе. Дополнительно для решения одномерного стационарного уравнения Шрёдингера с помощью теории возмущений были добавлены следующие функции:

* возмущенная потенциальная функция в строках 58-64;
* оператор возмущение 66-68.

В строках 440-546 реализовано решение одномерного стационарного уравнения Шрёдингера в рамках теории возмущений. В строках 460-468 определяется функция для получения матричного элемента оператора возмущения . В данной работе вычисление интегралов, необходимых для получения матричных элементов , осуществляется с использованием метода трапеций для численного нахождения интегралов, он так же был подробно разобрал в отчёте по первой лабораторной работе.

Используя формулы рассчитываются:

* первая поправка энергии в строках 472-474;
* вторая поправка энергии в 476-485;
* первая поправка собственной функции 487-491.

Результаты расчётов визуализируются с помощью Python библиотеки matplotlib в строках 504-540.

Результаты численных вычислений будут подробно проанализированы в следующей главе.

# Результаты численных экспериментов и их анализ

В данной работе необходимо было найти энергию основного состояния, с учётом поправок до второго порядка включительно и волновую функцию основного состояния, с учётом поправок первого порядка.

На рисунке 3 представлены результаты численного моделирования для возмущенной системы с волновой функции в первом приближении и рассчитанные методом пристрелки, оранжевая линия соответствует методу пристрелки для поиска собственной функции основного состояния, а голубая – теории возмущений. Смотря на графики видно, что функции, найденные двумя способами достаточно близки.

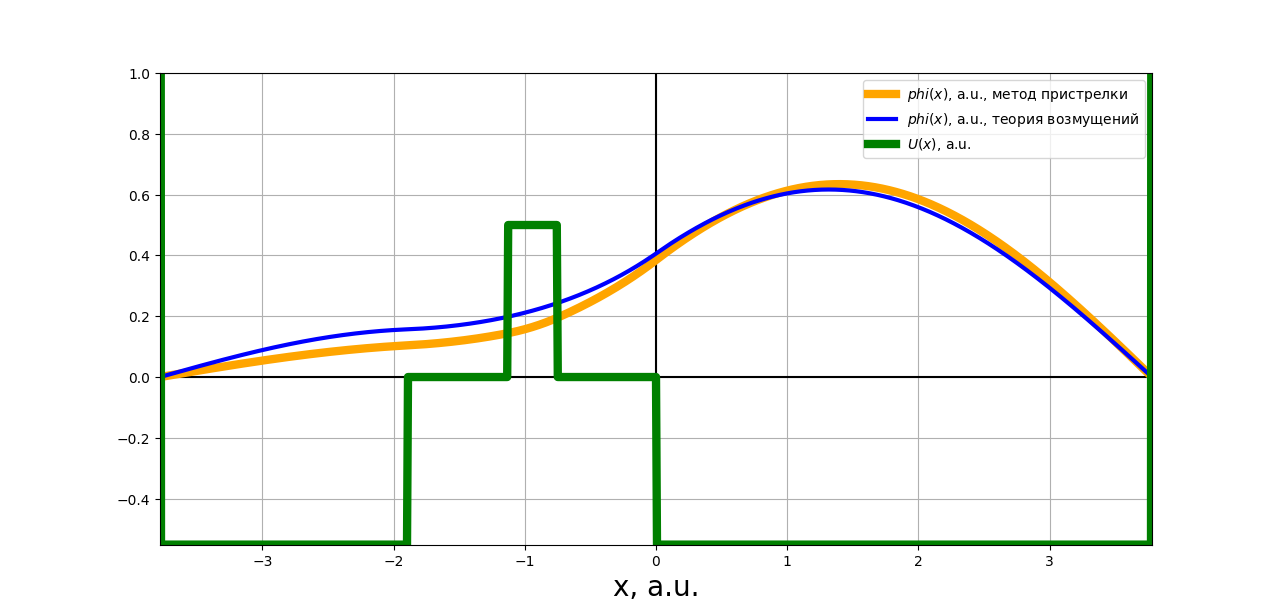


Рис. 3. Основное состояние

Вычисленная энергия основного состояния методом пристрелки:

и методом теории возмущений:

Видно, что значения совпадают до второго знака. Для вычисления энергии основного состояния с использованием теории возмущений были вычислены следующие поправки с использованием формул :

Из первой главы известно, что поправка второго порядка к энергии основного состояния не может быть положительной, видно, что результаты вычислительного эксперимента соотносятся с теоретической информацией.

# Список использованных источников

1. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: метод пристрелки. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.
2. Тимошенко Ю. К. Численное решение стационарного уравнения Шрёдингера: теория возмущений. Учебное пособие. Воронеж: Научная книга, 2019. 35 с.
3. Давыдов А. С. Квантовая механика. СПб.: БХВ-Петербург, 2011. 704 с.
4. Бизли Д. Python. Подробный справочник. СПб.: Символ-Плюс, 2010. 864 с.
5. Марчук А. Х. Введение в Python для студентов-астрономов. Методическое пособие. СПб.: СПбГУ, 2016. 49 с.
6. Доля П. Г. Введение в научный Python. Харьков: ХНУ, 2016. 265 с.

# Приложения

## Приложение 1 Компьютерный код

import matplotlib.pyplot as plt

from math import sqrt

######################################################################

### shooting begin

######################################################################

######################################################################

### bisection\_method

######################################################################

# метод деления отрезка пополам

def bisection\_method(f, # функция корень которой необходимо найти

begin:float, end:float, # начало и конец отрезка

# в котором находится корень

steps:int, # максимальное количество шагов алгоритма

eps:float): # точность приближения корня

l = begin

r = end

step = 0

while step < steps:

step += 1

center = 0.5 \* (r + l)

f\_l = f(l)

f\_c = f(center)

f\_r = f(r)

if abs(f\_c) < eps:

break

if f\_l \* f\_c < 0:

r = center

else: # f\_c \* f\_r < 0

l = center

return 0.5 \* (r + l)

######################################################################

### quantum\_mechanics

######################################################################

def v(x:float) -> float:

if (x > -half\_width and x < (-half\_width / 2.0)) or

(x > 0 and x < half\_width):

return -1

elif x >= -half\_width / 2.0 and x <= 0:

return 0

else:

return w

# Потенциальная функция невозмущенной системы

def U(x:float) -> float:

return v0 \* v(x)

# Потенциальная функция возмущенной системы

def U1(x:float) -> float:

if (x > (-half\_width / 2 + half\_width / 5) and

x < (-half\_width / 5)):

return U(x) + 0.5

else:

return U(x)

# возмущение

def V(x:float):

return U(x) - U1(x)

def U\_pr(x:float):

return U(x) + V(x)

SOLVE\_U = U

def q(e:float,

x:float) -> float:

return 2.0 \* (e - SOLVE\_U(x))

def compute\_q(x:list[float],

energy:float) -> list[float]:

return [q(energy, xi) for xi in x]

def derivative(function:list[float],

index:int,

eps:float) -> float:

der1 = function[index - 2] - function[index + 2]

der2 = function[index + 1] - function[index - 1]

return (

(der1 + 8.0 \* der2)

/ #-------------------

(12.0 \* eps)

)

# интегрирование вперёд

def forward\_integration(num\_intervals:int,

penultimate\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

forward = [float] \* (num\_points)

forward[0] = 0.0

forward[1] = penultimate\_approximation

c = step \*\* 2 / 12.0

for i in range(1, num\_intervals):

p1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* forward[i]

p2 = (1.0 + c \* q[i - 1]) \* forward[i - 1]

p3 = (1.0 + c \* q[i + 1])

forward[i + 1] = (

(p1 - p2)

/ #-----------

p3

)

return forward

# интегрирование назад

def backward\_integration(num\_intervals:int,

first\_approximation:float,

q:list[float],

step:float) -> list[float]:

num\_points = num\_intervals + 1

backward = [float] \* num\_points

backward[num\_intervals] = 0

backward[num\_intervals - 1] = first\_approximation

c = step \*\* 2 / 12.0

for i in range(num\_intervals - 1, 0, -1):

f1 = 2.0 \* (1.0 - 5.0 \* c \* q[i]) \* backward[i]

f2 = (1.0 + c \* q[i + 1]) \* backward[i + 1]

backward[i - 1] = (

(f1 - f2)

/ #--------------------

(1.0 + c \* q[i - 1])

)

return backward

# фукнция для нормировки forward

# и достижения равенства на узле сшивки

def normalization(forward:list[float],

backward:list[float],

connection:int):

# нормировка

norm = abs(max(forward, key = abs))

forward = list(map(lambda x: x / norm, forward))

# равенство на узле сшивки - connection

coef = forward[connection] / backward[connection]

backward = list(map(lambda x: coef \* x, backward))

return forward, backward

# функция возвращающая разницу производных на узле сшивки

def is\_close(forward:list[float],

backward:list[float],

connection:int,

eps:float) -> float:

return (derivative(forward, connection, eps)

- derivative(backward, connection, eps))

# функция вычисляет значения волновой функции вперёд и назад по известным данным

# и возвращающая разницу производных на узле сшивки

def is\_close\_energy(

energy:float, # значение энергии

x:list[float], # сетка

step:float, # шаг сетки

forward\_first\_approximation:float, # первое приближение

# для интегрирования вперёд

backward\_penultimate\_approximation:float, # n - 1 приближение

# для интегрирования назад

connection:int) -> float: # узел сшивки

num\_intervals = len(x) - 1

q = compute\_q(x, energy)

forward = forward\_integration(num\_intervals,

forward\_first\_approximation, q, step)

backward = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q, step)

forward, backward = normalization(forward, backward, connection)

return is\_close(forward, backward, connection, step)

# возвращет интервалы в которых находятся собственные значения оператора Гамильтона

def eigen\_value\_intervals(min\_energy:float,

energy\_step:float,

max\_energy\_value:float,

max\_energy\_count:int,

x:list[float],

step:float,

forward\_first\_approximation:float, # 1 приближение

# для интегрирования вперёд

backward\_penultimate\_approximation:float, # n - 1 приближения

# для интегрирования назад

connection:int) -> list[float]:

level = 0

sign = (1 if level % 2 == 0 else -1)

energy = min\_energy

prev\_close = is\_close\_energy(energy,

x,

step,

sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

intervals = []

intervals.append(energy)

while(energy < max\_energy\_value):

energy = energy + energy\_step

sign = (1 if level % 2 == 0 else -1)

close = is\_close\_energy(energy,

x,

step,

sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

if close \* prev\_close < 0:

prev\_close = close

intervals.append(energy)

intervals.append(energy)

level += 1

if(len(intervals) >= 2 \* max\_energy\_count):

break

else:

intervals[-1] = energy

intervals.pop()

return intervals

def integrate(f:list[float], step:float) -> float:

size = len(f)

sum = 0.0

for i in range(1, size - 1):

sum += f[i]

return step \* ((f[0] + f[size - 1]) / 2.0 + sum)

def quantum\_normalize(psi:list[float], step:float) -> list[float]:

density = list(map(lambda x: x \* x, psi))

c = integrate(density, step)

root = 1.0 / sqrt(c)

normalized = list(map(lambda x: root \* x, psi))

return normalized

def second\_derivation(f:list[float], step:float):

num\_points = len(f)

derivation = [float] \* num\_points

coef = 1 / (step \* step)

derivation[0] = coef \* (2 \* f[0] - 5 \* f[1] + 4 \* f[2] - f[3])

derivation[num\_points - 1] = (coef \*

(2 \* f[num\_points - 1]

- 5 \* f[num\_points - 2]

+ 4 \* f[num\_points - 3]

- f[num\_points - 4]))

for i in range(1, num\_points - 1):

derivation[i] = coef \* (f[i - 1] - 2 \* f[i] + f[i + 1])

return derivation

w = 10.0

v0 = 15 # Электронвольт

half\_width = 2.0 # Ангстрем

print(f"v0 = {v0} Электронвольт")

print(f"l = {half\_width} Ангстрем")

# Перевод величин в атомную систему единиц

v0 = v0 / 27.211 # атомных единиц

half\_width = half\_width / 0.5292 # бор

print(f"v0 = {v0} атомных единиц")

print(f"l = {half\_width} бор")

def main():

global SOLVE\_U

min\_energy = -v0

print("Минимальное значение потенциальной функции = {}".format(min\_energy))

print()

forward\_first\_approximation = 1.e-9

backward\_penultimate\_approximation = 1.e-9

begin = -half\_width

end = +half\_width

num\_intervals = 1000

num\_points = num\_intervals + 1

connection = 300

step = (end - begin) / num\_intervals

x = [begin + step \* i for i in range(num\_points)]

energy\_step = 0.01

max\_energy\_value = 1000

max\_energy\_count = 2

bisection\_method\_steps = 100

bisection\_method\_eps = 0.0001

######################################################################

### shooting perturbation

######################################################################

SOLVE\_U = U\_pr

intervals\_pr = eigen\_value\_intervals(min\_energy,

energy\_step,

max\_energy\_value,

max\_energy\_count,

x,

step,

forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

print("intervals\_pr: ", intervals\_pr)

energies\_pr = []

level\_pr = 0

f\_pr = lambda e : is\_close\_energy(e,

x,

step,

forward\_sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

for i in range(max\_energy\_count):

l = intervals\_pr[2 \* i]

r = intervals\_pr[2 \* i + 1]

forward\_sign = (1 if level\_pr % 2 == 0 else -1)

energy = bisection\_method(

f\_pr,

l, r,

bisection\_method\_steps,

bisection\_method\_eps)

energies\_pr.append(energy)

level\_pr += 1

print("energies\_pr: ", energies\_pr, " a.u.")

print()

q\_n0\_pr = compute\_q(x, energies\_pr[0])

forward\_n0\_pr = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n0\_pr, step)

backward\_n0\_pr = backward\_integration(num\_intervals,

backward\_penultimate\_approximation,

q\_n0\_pr, step)

forward\_n0\_pr, backward\_n0\_pr = normalization(forward\_n0\_pr,

backward\_n0\_pr, connection)

forward\_n0\_pr = quantum\_normalize(forward\_n0\_pr, step)

backward\_n0\_pr = quantum\_normalize(backward\_n0\_pr, step)

q\_n1\_pr = compute\_q(x, energies\_pr[1])

forward\_n1\_pr = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n1\_pr, step)

backward\_n1\_pr = backward\_integration(num\_intervals,

backward\_penultimate\_approximation,

q\_n1\_pr, step)

forward\_n1\_pr, backward\_n1\_pr = normalization(forward\_n1\_pr,

backward\_n1\_pr, connection)

forward\_n1\_pr = quantum\_normalize(forward\_n1\_pr, step)

backward\_n1\_pr = quantum\_normalize(backward\_n1\_pr, step)

######################################################################

### shooting idealized

######################################################################

SOLVE\_U = U

intervals = eigen\_value\_intervals(min\_energy,

energy\_step,

max\_energy\_value,

max\_energy\_count,

x,

step,

forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

print("intervals: ", intervals)

energies = []

level = 0

f = lambda e : is\_close\_energy(e,

x,

step,

forward\_sign \* forward\_first\_approximation,

backward\_penultimate\_approximation,

connection)

for i in range(max\_energy\_count):

l = intervals[2 \* i]

r = intervals[2 \* i + 1]

forward\_sign = (1 if level % 2 == 0 else -1)

energy = bisection\_method(

f,

l, r,

bisection\_method\_steps,

bisection\_method\_eps)

energies.append(energy)

level += 1

print("energies: ", energies, " a.u.")

print()

q\_n0 = compute\_q(x, energies[0])

forward\_n0 = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n0, step)

backward\_n0 = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q\_n0, step)

forward\_n0, backward\_n0 = normalization(forward\_n0,

backward\_n0, connection)

forward\_n0 = quantum\_normalize(forward\_n0, step)

backward\_n0 = quantum\_normalize(backward\_n0, step)

q\_n1 = compute\_q(x, energies[1])

forward\_n1 = forward\_integration(num\_intervals,

forward\_first\_approximation, q\_n1, step)

backward\_n1 = backward\_integration(num\_intervals,

backward\_penultimate\_approximation, q\_n1, step)

forward\_n1, backward\_n1 = normalization(forward\_n1,

backward\_n1, connection)

forward\_n1 = quantum\_normalize(forward\_n1, step)

backward\_n1 = quantum\_normalize(backward\_n1, step)

######################################################################

### shooting end

######################################################################

######################################################################

### perturbation begin

######################################################################

# энергии

# energies

# волновые функциий

wave = [forward\_n0, forward\_n1]

# Потенциальная функция невозмущенной системы

U0 = U

# Потенциальная функция возмущенной системы

# U1

# возмущение

# V

def matrix\_V(n:int, m:int):

v\_n\_m = [wave[n][i] \* v[i] \* wave[m][i]

for i in range(num\_points)]

return integrate(v\_n\_m, step)

x = [begin + step \* i for i in range(num\_points)]

v = [V(xi) for xi in x]

u0 = [U0(xi) for xi in x]

u1 = [U1(xi) for xi in x]

energy\_index = 0

# первая поправка энергии

first\_order\_correction\_e = matrix\_V(energy\_index, energy\_index)

print("первая поправка энергии=", first\_order\_correction\_e)

# вторая поправка энергии

secont\_order\_correction\_e = 0

for i in range(len(energies)):

if(i != energy\_index):

secont\_order\_correction\_e += (

abs(matrix\_V(energy\_index, i)) \*\* 2

/ #--------------------------------------

(energies[energy\_index] - energies[i])

)

print("первая поправка энергии=", secont\_order\_correction\_e)

# первая поправка собственной функции

const = matrix\_V(0, 1) / (energies[0] - energies[1])

mul = [const \* w for w in wave[1]]

first\_order\_correction\_w = [wave[0][i] + mul[i]

for i in range(len(wave[1]))]

# вывод

energy = energies[0]

print("E\_0 = ", energy)

energy += first\_order\_correction\_e

print("E\_0 = ", energy, " 1-ое приближение, ", "поправка =", first\_order\_correction\_e)

energy += secont\_order\_correction\_e

print("E\_0 = ", energy, " 2-ое приближение, ", "поправка =", secont\_order\_correction\_e)

# Графики

plt.axis([begin, end, min\_energy, v0])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.ylabel("$U^0(x)$, a.u.", fontsize = 20, color = "k")

plt.plot(x, u0, linewidth = 6, color = "red", label = "$U^0(x)$")

plt.legend()

plt.show()

plt.axis([begin, end, min\_energy, v0])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.ylabel("U(x), a.u.", fontsize = 20, color = "k")

plt.plot(x, u1, linewidth = 6, color = "red", label = "U(x)")

plt.legend()

plt.show()

plt.axis([begin, end, min\_energy, v0 if v0 > 1 else 1])

plt.grid(True)

plt.axhline(0, color='black')

plt.axvline(0, color='black')

plt.xlabel("x, B", fontsize = 20, color = "k")

plt.plot(x, forward\_n0\_pr, linewidth = 6,

color = "orange",

label = "$phi(x)$, a.u., метод пристрелки")

plt.plot(x, first\_order\_correction\_w, linewidth = 3,

color = "blue",

label = "$phi(x)$, a.u., теория возмущений")

plt.plot(x, u1, linewidth = 6,

color = "green",

label = "$U(x)$, a.u.")

plt.legend()

plt.show()

######################################################################

### perturbation end

######################################################################

main()